UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Mobilní čtečka DataMatrix kódu

Matěj Musil

Bakalářská práce
2014

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2013/2014**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Matěj Musil**
Osobní číslo:	**I11001**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Mobilní čtečka DataMatrix kódu**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem práce je vytvořit mobilní aplikaci pro čtení DataMatrix kódů a jejich následné zpracování.
Teoretická část:
Popis jazyka Java s důrazem na platformu Android. Popis technologií, které přímo souvisí s aplikací. Zabezpečení přenosů mezi OS Android a databázovým systémem. Popis a vhodný výběr API pro čtení DataMatrix kódů.
Praktická část:
Vyhotovení mobilní aplikace pro práci s DataMatrix kódy používané firmou Explosia a.s. a následné zpracování dat, jejich katalogizace a export do formátu PDF. Zakomponování synchronizace databáze mobilního zařízení se serverem.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

UJBÁNYAI, Miroslav. Programujeme pro Android. Vyd. 1. Praha: Grada, 2012, 187 s. Průvodce (Grada). ISBN 978-80-247-3995-3.
MURPHY, Mark L. Android 2: průvodce programováním mobilních aplikací. Vyd. 1. Brno: Computer Press, 2011, 375 s. Průvodce (Grada). ISBN 978-80-251-3194-7.
HEROUT, Pavel. Java a XML: průvodce programováním mobilních aplikací. 1. vyd. České Budějovice: Kopp, 2007, 313 s. Průvodce (Grada). ISBN 978-80-7232-307-4.X

Vedoucí bakalářské práce: **Ing. Zdeněk Šilar**
Katedra informačních technologií

Datum zadání bakalářské práce: **20. prosince 2013**
Termín odevzdání bakalářské práce: **9. května 2014**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2014

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 9. 5. 2014 Matěj Musil

Poděkování

V první řadě bych chtěl poděkovat Karlu Linkovi za nápad a pomoc s testováním aplikace. Dále bych rád poděkoval Ing. Zdeňku Šilarovi, Ph.D. za pomoc při zpracování aplikace i celé bakalářské práce a vstřícný přístup. Děkuji přátelům i rodině za podporu a trpělivost. A v neposlední řadě patří velký dík mé sestře Mgr. Kateřině Musilové za nekončící korekturní práce.

Anotace

Cílem práce je vytvoření mobilní čtečky DataMatrix kódů pro operační systém Android. Teoretická část práce se zabývá porovnáním existujících systémů na trhu, dále popisem knihoven pro čtení a dekódování DataMatrix kódů, popisem operačního systému Android, databází SQLite, jazyka XML a knihovny Zxing. V neposlední řadě také popisem zabezpečení databáze na OS Android. Implementační část pojednává o realizaci aplikace. Rozebírá použití knihovny Zxing, strukturu databáze, tříd a Android komponent. Dále jsou popsány problémy katalogizace dat a jejich úpravy. Závěr práce pojednává o možnostech rozšíření a srovnání s konkurenčními produkty.

Klíčová slova

DataMatrix, Android, Java, SQLite, XML, Zxing

Title

Mobile reader for DataMatrix code

Annotation

Aim of the bachelor thesis is to create mobile reader of DataMatrix code for Android operating system. The theoretical part focuses on comparison of existing systems on the market. Furthermore, it deals with different libraries for decoding and reading Datamatrix code, SQLite database, XML technology and library Zxing. In the final part , a description of database security on Android OS. The implementation section deals with the realization of the application. There is described of the usage Zxing library, structure of database, classes and Android component. Furthermore, the problem of cataloging data and its editing options. Final part focuses on the possibilities of extension and comparison with competition.

Keywords

DataMatrix, Android, Java, SQLite, XML, Zxing

Obsah

Seznam zkratek

OS	Operační systém
XML	eXtensible Markup Language
ECC	Error correction codes
ER	Entity Relationship
DVM	Dalvik Virtual Machine
VM	Virtual Machine
UML	Unified Modeling Language
SDK	Software Development Kit
API	Application Programming Interface
SQL	Structured Query Language
GPS	Global Positioning System
PDF	Portable Document Format
QR	Quick Response
W3C	World Wide Web Consortium
RSS	Reduce Space Symbology
ITF	Interleaved Two of Five
EAN	European Article Number
SD	Secure Digital
UPC	Universal Product Code

Seznam obrázků

Úvod

Výroba, skladování a přeprava jakéhokoliv zboží vyžadují jasné označení. V minulosti plně dostačovala kombinace sériového čísla s názvem produktu. Nyní se sériově vyráběné výrobky označují, mimo výše zmíněnou kombinací označovacích prvků, také strojově čitelným kódem, například „čárkovým“ kódem, který se typicky nachází na výrobcích v supermarketech. Použití těchto kódů je nám demonstrováno u pokladny, kde pracovník obchodu pomocí čtečky načte informace o výrobku a pokladní program vyhledá v databázi cenu produktu a následně nám tuto cenu přičte k účtu.

Stejný princip využívají firmy, které se zabývají výrobou výbušnin. Ovšem v tomto odvětví průmyslu se používá speciální označení typu kódu – DataMatrix. Jedná se o dvoudimenzionální kód, který se skládá z černých a bílých buněk, ve kterých je zakódován text nebo číslo. Tyto kódy jsou z pravidla čtvercového tvaru a na dvou bocích se nachází černá čára (připomínající písmeno L). Tato čára slouží pro zjištění orientace kódu a jeho následné rozkódování. Kódy můžeme nalézt v různých velikostech, od 10x10 po 144x144 (v nejnovější verzi ECC 200), první číslo označuje počet řádků a druhé počet buněk na řádku. DataMatrix kódy se nejčastěji používají v průmyslu a korporátní sféře, na rozdíl od běžně rozšířených QR kódů.

Výbušniny jsou samozřejmě velmi nebezpečné, a proto se je rozhodla Policie ČR důkladně sledovat. Z tohoto důvodu musí být každá výbušnina (krabice, paleta) označena unikátním kódem a musí se vést evidence o pohybech výbušnin. Např. kde byla výbušnina vyrobena, kým byla přepravovaná a kam, následně kde byla použita. Tento fakt přináší problém s vedením evidence a zapisováním do ní. Kódy označující výbušniny jsou v drtivé většině velmi obsáhle a nesou, mimo unikátní číslo, i další informace. Pro přepravce a střelmistry je opisování těchto kódů velmi zdlouhavé, obzvláště při naskladňování a vyskladňování zboží, protože i malý odběratel odebírá 5-10 palet výbušnin ze skladu.

Cílem této práce je navržení a implementování programu pro mobilní operační systém Android. Tento program bude dekódovat DataMatrix kódy na výbušninách a poté je ukládat od databáze. Záměrem aplikace je zrychlení přijímání a odepisování výbušnin pro menší firmy.

Na samotnou aplikaci jsou kladeny následující požadavky: čtení kódů pomocí vestavěného fotoaparátu, uložení kódů s příslušnými údaji (o jaký produkt se jedná, místo nákupu), vytváření databáze kódů a jejich případný export do dokumentu PDF.

1 Existující systémy

V praxi obdobné systémy existují a jsou denně používány. Čtečky kódů se nacházejí na výrobních linkách, ve skladech i u velkých přepravců. U pásové výroby je běžné, že linka obsahuje snímač kódů zabudovaný a napojený na centrální systém. Skladníci i větší dopravci mívají přenosné terminály, které jsou také napojeny na centrální systém.

Jedním z těchto systémů je Motorola MC9500-K (viz Obrázek 1). Jedná se o mobilní terminál, který do sebe integruje velké množství senzorů. Tento terminál tvoří 3,7 palcový displej a klávesnice na přední straně, na zadní straně se nacházejí baterie a fotoaparát. A nakonec na přední straně se nachází čip pro čtení jednodimenzionálních i dvoudimenzionálních kódů. Tělo je ergometricky tvarováno, je odolné vůči pádu z výšky do 1,8m a splňuje třídu krytí IP67. Mezi vybavení terminálu patří také GPS modul, Bluetooth modul i WiFi modul. O uživatelské rozhraní se stará starší operační systém Microsoft Windows Mobile 6.1 (nebo 6.5, záleží na přesném modelu). Pro načítání kódů lze použít aplikaci dodávanou jakou součást operačního systému nebo ji nahradit jinou aplikací kompatibilní se systémem Windows Mobile. K tomuto zařízení je možné dokoupit velké množství příslušenství, které výrazně zlepší práci se systémem, například pistolovou rukověť, přídavné baterie, dokovací stanice apod. Při práci na lomech pracovníci ocení zmíněné krytí IP67[1]. Tento terminál je bohužel finančně velmi nákladný. Základní verze přístroje se základním příslušenstvím překračuje částku 60 tisíc korun. Pokud by firma potřebovala ještě složitější software na uchovávání informací o kódech, jako je místo a datum sejmutí, musí si připravit další 10 tisíc korun. Takovéto částky jsou pro většinu malých firem neakceptovatelné a musí proto hledat jiná řešení.

Obrázek 1 - Motorola MC9500 [10]

Na začátku roku 2014 se na trhu začal objevovat nový systém. Jedná se o tablet s operačním systém Android a externím skenerem připojeným pomocí kabelu USB. Tablet vyrábí čínská firma a na těle má plnohodnotný USB konektor pro připojení čtečky. Aplikace nabízí vše potřebné pro nastavení a uložení kódů. Tento systém vyjde na 15 tisíc korun. Uvedené řešení je úplnou novinkou a tento nový produkt je neustále zlepšován.

[1] Stupeň krytí zařízení. Chrání před vniknutím prachu a ponořením na 30 min do hloubky 1 metr.

2 Java v systému Android

Většina aplikací pro zařízení Android je napsaná v jazyce Java, ale existují velké rozdíly mezi Java API (nejčastěji použita na počítačích) a Android API. Na platformě Android neexistuje Java Virtual Machine. Třídy Java jsou kompilovány do proprietárního bitového kódu, který běží na virtuálním stroji Dalvik. Dalvik VM je založen na registrové architektuře, na rozdíl od zásobníkové architektury Java VM. Bitové kódy pro Dlavik VM jsou rozdílné od bitových kódů Java, proto není možné knihovny Java (jar soubor) vkládat do programu, který je určen pro OS Android. Nicméně většina prostředí pro vývoj aplikace dokáže takovouto knihovnu připojit pomocí indexovacího souboru dex. [1], [2], [4]

Dalvik VM se liší od jiných standardních virtuálních strojů. Mezi jeho charakteristické rozdíly patří:

- menší spotřeba operační paměti,
- Constant pool, místo pro uložení dat proměnných, byl rozšířen na 32bitů,
- 16bitové instrukce, které umožňují práci s proměnnými. Standardní bitový kód Java používá 8bitové operace, což zapříčiní přesun místních proměnných do zásobníku jinou operací.

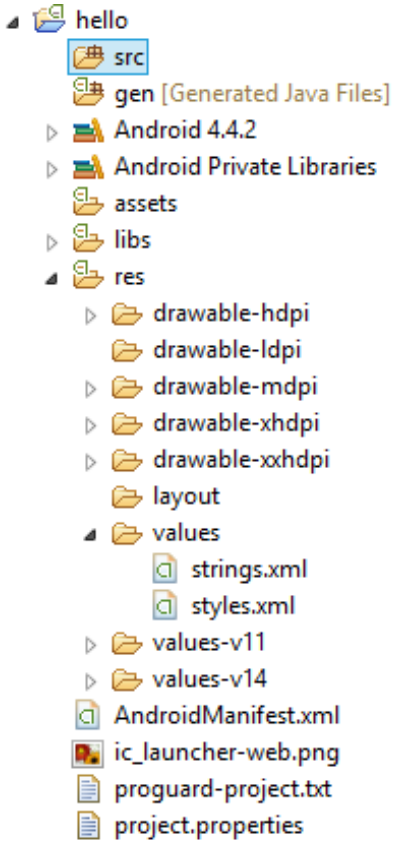
Při programování aplikací pro operační systém Android se používají kromě programovacího jazyka Java také XML soubory. XML soubory zastávají mnoho funkcí, slouží pro uchování textových řetězců (usnadnění překladu aplikace), informací o aplikaci, rozložení komponent při zobrazování atd.

Standardní Android projekt psaný v jazyce Java obsahuje adresářovou strukturu jako na Obrázek 2. V adresáři root se nachází hlavní popisovací soubor AndroidManifest.xml. V tomto souboru se nachází soupis aktivit, práv aplikace a dalších důležitých základních informací. V adresáři src se nacházejí zdrojové soubory Java. Adresář assets slouží pro uložení souborů pro chod aplikace. Jakýkoliv soubor uložený do této složky se přenese do mobilního telefonu a je aplikaci k dispozici. Dalším adresářem je res (zkratka resorces), v tomto adresáři se nacházejí další adresáře pro ikony, obrázky v různých rozlišeních. Mezi další adresáře v této složce patří složky values pro uchování XML souborů s daty o textových řetězcích, daty v polích atd. Poslední složkou je adresář layout. Do této složky patří XML soubory popisující rozložení komponent na obrazovce dané Aktivity/Fragmentu.

Operační systém Android nerozlišuje zařízení podle rozlišení obrazovky, ale podle hustoty pixelů, a to na čtyři základní skupiny:

- ldpi – nízká hustota 120 dpi,
- mdpi – střední hustota 160 dpi,
- hdpi - vysoká hustota 240 dpi,
- xhdpi – velmi vysoká hustota 320 dpi.

V projektu se nachází speciální adresář gen, do tohoto adresáře se automaticky generují třídy při programování. Generovaná data se zpřístupňují pomocí třídy `R`. V této třídě se nacházejí data z XML souborů z celého projektu. Přes tuto třídu tedy lze přistupovat ke komponentám, řetězcům i ostatním datům v XML souborech.[5]



Obrázek 2- Adresářová struktura projektu

3 Knihovny DataMatrix kódů

Pro správný chod a efektivitu aplikace je zapotřebí kvalitní a rychlá knihovna pro přečtení a následné dekódování informací z kódu DataMatrix. V dnešní době již existuje velké množství knihoven, které spolupracují i s operačním systémem Android, ovšem jsou mezi nimi velké rozdíly.

Na knihovnu potřebnou pro chod aplikace jsou kladeny specifické požadavky, podle kterých byla učiněna volba. Mezi tyto požadavky patří:

- cena,
- možnosti použití,
- rychlost zpracování,
- dokumentace,
- rozšiřování projektu.

3.1 Knihovna i-nigma SDK

Jedná se o komerční knihovnu vyvíjenou firmou 3GVision. Mezi klíčové vlastnosti patří rychlost zpracování kódů. Na reklamních videích firma demonstruje rychlost čtení na zhruba 20 kódech, které byly různého typů (čárové kódy, QR i DataMatrix). Pracovník přejížděl telefonem ve velké rychlosti přes všechny a ty se bez nejmenšího zaváhání načetly do paměti telefonu. Jelikož se jedná o komerční produkt, je dostupná široká dokumentace a technická podpora. Bohužel cena této knihovny je pro zamýšlené použití příliš vysoká, po delší komunikaci s firmou byla cena stanovena na 20 000Kč (hrubý přepočet z amerického dolaru) za rok, což je cena zcela neakceptovatelná pro zamýšlené vyžití navržené aplikace.

Knihovna se vyskytuje v aplikacích, které se běžně používají a jsou zdarma. Příkladem může mobilní aplikace na skenování kódu od firmy Seznam (Seznam.cz QR čtečka). Dalšími klienty firmy jsou banky, samozřejmě banky neuveřejňují podrobnosti o knihovnách použitých v jejich aplikacích. Tedy lze pouze spekulovat o tom zda, používají právě tuto knihovnu, nicméně firma 3GVision se mimo jiné specializuje na práci s bankami. [6]

3.2 Zxing SDK

Jedná se o open source[2] projekt pod licencí Apache Software License 2.0, což dovoluje komerční použití, pokud bude zobrazena původní licence ve finálním produktu. Knihovna je implementována v jazyku Java a nabízí i portaci do jiných jazyků (C++, Objective C). Co se týče rychlosti zpracování dat, je knihovna pomalejší než i-nigma. Srovnatelné výsledky dosahuje při ideálních podmínka (světlo, kvalita snímače a kódu), ovšem v horších podmínkách zaostává. Nejvíce problémů nastává při snímání kódů pod úhlem (na

[2] Otevřený zdrojový kód. Reprezentuje snadnou licenční i technickou dostupnost.

čemž má kolektiv v plánu nadále pracovat) a při snímání kódů, které odráží světlo. Komunita aktivně přidává další funkcionalitu. Dokumentace je na rozumné úrovni a na různých diskuzních fórech je řada doplňujících informací.

Komunita vydala i vlastní aplikaci pod jménem Barcode Scanner, kterou lze volně stáhnout na Google play. Vývojáři nabízí široké možnosti různých formátů, které lze skenovat. Mezi nimi se nachází i DataMatrix kód. Tato knihovna je nejideálnější pro použití za daných podmínek v roce 2014. [7]

3.3 ZBar bar

Knihovna je open source a šířená pod licencí GNU[3] LGPL 2.1, což umožnuje bezproblémové šíření i úpravu zdrojových kódů. Snímání kódů a jejich následné dekódování probíhá nejpomaleji a s nejmenší přesností. Oproti knihovně Zxing má Api lepší a přehlednější dokumentaci. Bohužel tato knihovna nedokáže přečíst DataMatrix kódy a její vývoj za poslední dva roky zásadně nepokročil.

Vývojáři knihovny ZBar nedodali žádnou aplikaci pro OS Android, na které by bylo snazší vyzkoušení funkčnosti. Řešením je napsat aplikaci na vyzkoušení nebo provést testy na telefonech firmy Apple iPhone. Komunita okolo Api je stále aktivní a diskuzní fóra jsou plná konverzací, bohužel hlavní konverzace se týká vývoje pro iOS[4]. [8]

[3] GNU General Public License. Licence pro svobodný sofware.
[4] Operační systém společnosti Apple určený pro produkty výhradně společnosti Apple.

4 Použité technologie

4.1 OS Android

Operační systém Android je postaven na linuxovém jádře, které implementuje důležitou vlastnost celého projektu - přenositelnost. Systém je díky této vlastnosti možné aplikovat na zařízení, která jsou osazena různými chipsety nebo disponují odlišnými velikostmi displejů. Základy projektu Android vznikly v roce 2003, o dva roky později tento projekt zakoupila společnost Google. Během několika málo let se dokázal dostat do popředí světového zájmu a nakonec ovládl trh s operačními systémy mobilních telefonů.[1], [2]

4.1.1 Základní komponenty

Aplikace určené pro operační systém Android se skládají ze čtyř základních prvků:

- Aktivity – vyjadřuje jednu obrazovku. Komponenta obsahuje grafické uživatelské rozhraní. Aplikace obsahují typicky větší množství aktivit, ve kterých jsou umístěny jednotlivé formuláře, texty, obrázky atd. Aktivita se může nacházet ve více stavech, systém těchto stavů se nazývá životní cyklus. Mezi hlavní stavy aktivity patří: start (inicializace aktivity), running (Activity je zobrazeno na displeji a může mít interakci s uživatelem), shut down (activita již nevyužívá žádnou paměť).

- Service – komponenta určená pro funkce, které pracují po delší dobu. Jedná se o proces běžící na pozadí, neosahuje tedy grafické uživatelské rozhraní.

- Content providers – prvek určený pro sdílení dat mezi různými aplikacemi. Komponenta je velmi podobná databázi (data se ukládají do tabulek).

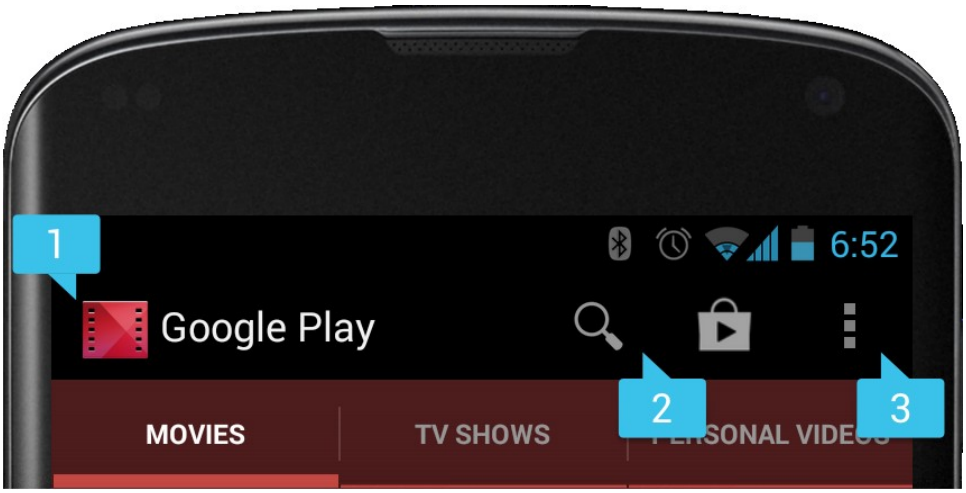
- Broadcast reciever – komponenta sloužící pro zachytávání globálních oznámení, např. nízký stav baterie nebo přijmutí SMS zprávy. Prvek neobsahuje grafické uživatelské rozhraní.

4.1.2 Nové standardy

Firma Google se snaží nastavit pravidla (standardy) ve vzhledu aplikací a jejich ovládacích prvcích. V poslední plné verzi (4. x) společnost vydala inovovanou knihovnu, která obsahuje nové prvky pro ovládání aplikace a zobrazení obsahu. Jedná se mimo jiné o Fragmenty a ActionBar menu.[9]

Fragmenty celkově označují nový přístup ke tvorbě uživatelského rozhraní, kdy mezi Activity a View vstupuje ještě jedna vrstva, a to Fragment. Takový Fragment má za úkol „obalit" nějaký funkční celek, tedy například seznam poznámek, zobrazení jedné poznámky (titulek a text) či formulář přidávající novou poznámku, a to jak potřebná View, tak metody s nimi související. Fragment obalující seznam poznámek se postará například o naplnění ListView daty či zobrazení kontextového menu nad položkou.

Action bar je funkce horní lišty aplikace, ukázka rozložení na Obrázek 3 – Actionbar. Tato vlastnost poskytne uživateli ovládat aplikaci a zpřístupní snadnou navigaci. Protože toto ovládání funguje ve většině dnešních aplikací, uživateli nedělá žádný problém adaptovat se na novou aplikaci



Obrázek 3 – Actionbar [11]

4.2 SQLite databáze

SQLite je odlehčená verze databázového systému SQL. Tento systém je masivně využíván u aplikací, které nepotřebují rozsáhlý databázový systém. Systém je využíván pro jeho rychlost a šetrnost k paměti. Je tvořen jedinou knihovnou, což je velmi praktické oproti často používaným způsobům klient-server. SQLite využívá jazyk SQL podle standardu SQL-92. Velkým problémem SQLite je neexistence typové kontroly, systém bez jakéhokoliv upozornění dovoluje například vkládání datového typu `numeric` do pole, které je typu `text`.

4.2.1 Vytvoření databáze

V operačním systému Android je poskytnuta třída `SQLiteOpenHelper`, která se stará o veškerou komunikaci s databází (vytváření, aktualizace, mazání databáze). Pro bezpečnější chod aplikace i databáze se doporučuje tuto třídu obalit vlastní metodou. `SQLiteOpenHelper` nabízí následující nejdůležitější metody:

- `onCreate` – je metoda sloužící pro vytváření a naplnění tabulek v databázi, tato metoda je použita při neexistenci databáze,
- `onUpgrade` – je určena pro aktualizaci databáze, metoda je aktivována při zjištění novější verze databáze (vyšší číslo verze),
- `onOpen` – je metoda, která se aktivuje po každém novém otevření databáze.

4.2.2 Práce s tabulkami

Při prvním vytvoření databáze se volá přetížená metoda `onCreate`, v níž jsou uvedeny příslušné SQL příkaz, které specifikují strukturu tabulek vytvářené databáze. Pro vlastní práci s daty databáze je třeba získat instanci třídy `SQLiteDatabase`, a to buď pomocí metody `getWritableDatabase`, nebo `getReadableDatabase`. Tyto metody vracejí instanci mající oprávnění k přístupu k zápisu, nebo pouze ke čtení. Pro práci s tabulkami se využívá metoda `sqlexec`.

4.2.3 Zobrazení dat

Vyhledávání dat je možné realizovat třemi různými způsoby, a to pomocí metod: `rawQuery`, `query` a třídy `SQLiteQueryBuilder`. První varianta představuje nebezpečnou metodu pro získání dat, do parametru vkládáme pouze SQL dotaz. Druhou variantou je bezpečná metoda `query`, obsahuje mnoho parametrů pro sestavení dotazu. Poslední způsob je vytvoření speciální třídy `SQLiteQueryBuilder`, pomocí této třídy je možné vytvořit i složitý dotaz (například dotaz, který spojuje více tabulek do jednoho dotazu). Všechny tři varianty vrací třídu `Cursor`.

`Cursor` je speciální třída reprezentující řádky, které se navrátily z dotazu. Práce s touto výslednou třídou připomíná práci s iterátorem. Po získání instance z databáze se musí objekt nastavit na začátek (metoda `moveToFist`), následně se řádky prochází pomocí metody `moveToNext`. Konec řádků signalizuje metoda `isAfterLast`. Na aktuálním řádku se získávají hodnoty sloupců pomocí metod začínající slovem `get` a následuje požadovaný typ proměnné, např. `getString` nebo `getInt`.

4.2.4 Vkládání dat

Pro vložení dat se využívá zabezpečená metoda `insert`. Jako parametr přijímá metoda třídu `ContentValues`, do této třídy se musí předpřipravit data. Vkládání dat probíhá s pomocí metody `put`.

4.2.5 Úprava dat

Úprava dat probíhá naprosto stejným způsobem jako vkládání dat, s využitím rozdílné metody (`update`). Tato metoda přijímá také třídu `ContentValues`, do které se musí předpřipravit data.

4.2.6 Odebírání dat

Odebírání zabezpečuje metoda `delete`. Metoda přijímá parametry obsahující název tabulky, ze které bude probíhat mazání, řetězec obsahující klauzuli `WHERE` a hodnoty, které se budou vkládat do klauzule `WHERE`.

4.3 Knihovna Zxing

Knihovna umožňující načítání a dekódování jednoho nebo dvou dimenzionálních kódů. Podporuje širokou škálu kódů: UPC-A, Code 39, QR Code, UPC-E, Code 93, Data Matrix, EAN-8, Code 128, Aztec (beta), EAN-13, Codabar, PDF 417 (beta), ITF, RSS-14, RSS-Expanded. API pro OS Android obsahuje i ovládací třídy pro fotoaparát a zobrazení výsledků. Nejnižší verze OS Android je verze HONEYCOMB (3.0.0). Jádro knihovny tvoří zdrojové kódy v jazyce Java. Toto jádro zajišťuje získání dat z obrázku zpět na textový řetězec. Knihovnu lze stáhnou jako kompletní řešení pro různé platformy (Android, iOS, Java, C++ atd.), nebo je možné získat kompletní zdrojové kódy a ty následně upravit na vlastní řešení. Poslední variantou je opatřit pouze jádro jako knihovnu Java (jar soubor) a implementovat třídy pro práci s fotoaparátem a uživatelským rozhraním.

Knihovnu pro operační systém Android je možné použít jako samostatnou aplikaci, která bude v nabídce programů viditelná a spustitelná zvlášť, nebo použít předpřipravené knihovny. A nakonec je zde možnost použít přímo zdrojové kódy, upravit je podle potřeby aplikace a následně kompilovat jako knihovnu.

4.4 XML technologie

eXtensible Markup Language (zkráceně XML) je jedním z nejrozšířenějších obecných značkovacích jazyků. XML umožňuje vytváření vlastních tagů (značek). Tímto přístupem nabízí programátorovi navrhnout ideální strukturu dokumentu. Značkovací jazyk XML využívá především výměnu dat mezi aplikacemi (klientem a serverem), nebo pro transformaci dat jiného formátu. [3]

XML bylo vyvinuto konsorciem W3C (World Wide Web Consortium). Konsorcium poskytuje tuto technologii zcela zdarma, tím pádem může jakýkoliv vývojář implementovat podporu XML do svého projektu. XML využívá implicitně znakovou sadu Unicode (případně ISO 10646), tato znaková sada zajišťuje podporu většiny světových jazyků.

Každý dokument by měl splňovat následující závazná pravidla:

- na začátku dokumentu musí být tag obalující celý zbytek dokumentu,
- elementy musí být párové a musí začínat ve tvaru `<jmeno_elementu>` a končit ve formátu `</jmeno_elementu>`,
- elementy se nesmí křížit, což znamená, že nesmí být otevřen nový tag, který by končil za ukončením předchozího.

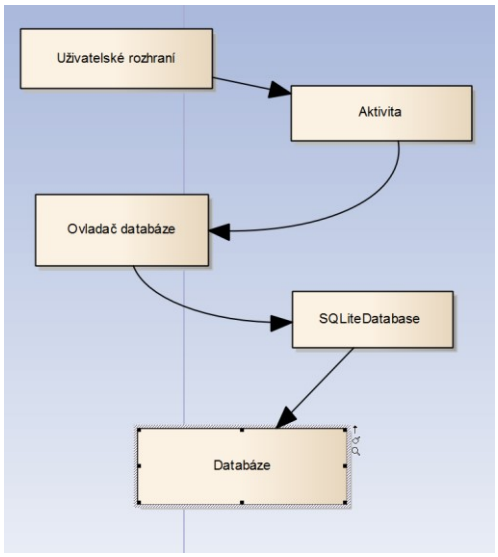
5 Zabezpečení databáze

Jedním z problémů databází dnešní doby je ochrana dat před zneužitím nebo poškozením. Z tohoto důvodu musí vývojáři aplikací chránit komunikaci a práci s databází. Mezi základní útoky patří: SQL injection, odposlech dat a neoprávněný přístup k databázi.

Aplikace v systému Android pracují s vestavěným databázovým systém SQLite. Tento systém vytváří pro každou databázi nový soubor do složky aplikace. OS Android vytváří soubory databáze při instalaci aplikace, a to do adresáře `DATA/data/JMÉNO_APLIKACE /databases/`. Do těchto složek nemá uživatel přístup, ovládání složky je možné pouze s oprávněním root. Rozšířené systémové oprávnění root získá uživatel pouze po zásadní úpravě systému. Pokud uživatel získá administrátorská oprávnění, nic mu nebrání k volnému přístupu souborů databáze.

Pro práci s databází se využívá třída *`SQLiteDatabase`* z knihovny Android API. Tato obsahuje mnoho metod, které jsou schopny pracovat s databází. Mezi základní metodu, kterou lze využít pro práci s databází, patří *`rawQuery`*. S touto metodou lze data získat (příkaz `select`), vkládat (příkaz insert), upravovat (příkaz update) i mazat (příkaz `delete`). Do této metody se vkládá SQL dotaz „na přímo“, tento způsob práce s databází je velmi náchylný k útoku. Jedná se o útok typu SQL injection, při tomto útoky uživatel zadá do jednoho pole určeného pro zadávání dat SQL příkaz (například pro smazání dat). Výsledkem je poškození nebo zneužití dat z databáze. Aby se předešlo tomuto útoku, je nutné použití bezpečných metod, které třída obsahuje. Třída nabízí zvláštní metody pro každou akci, do met se vloží postupně data a metoda sama vytvoří SQL příkaz. Mezi tyto metody patří: `query` (získání dat), `insert` (vkládání), `delete` (mazání záznamů) atd.

Pro dosažení vyšší bezpečnosti navrhli vývojáři operačního systému Android vícevrstvý model pro komunikaci s databází. Tento návrhový vzor doporučuje programátorům aplikací třídu, pro kterou zajistí kompletní komunikaci s vnitřními třídami Android API. Návrhový model je zobrazen na Obrázek 4.



Obrázek 4 - Návrhový vzor databáze

6 Implementace aplikace

Aplikace se skládá ze dvou základních Aktivit, zobrazovacích Fragmentů, datových objektů, obslužných adaptérů a handlerů řídících chod databáze a GPS modulu. Systém je navržen na operační systém Android od verze 4.0. V této verzi byly poprvé představeny Fragmenty pro mobilní telefony. Tuto funkcionalitu zajišťuje knihovna android-support-v4. Pro čtení je vložena a upravena knihovna Zxing.

Projekt byl celý tvořen v programovacím prostředí IntelliJ IDEA 13.1.1 a byl rozdělen do dvou samostatných modulů. První modul obsahuje uživatelské rozhraní, databázi a logiku zpracování informací. Druhý modul je knihovna Zxing, ve které proběhly úpravy pro optimálnější chod aplikace. Vývojové prostředí umožňuje přímé sledování konzole připojeného zařízení a v posledních verzích i přímou komunikaci s databází aplikace v telefonu.

6.1 API Zxing

Knihovna Zxing je po stažení připravena ke kompilaci a nasazení bez dalších úprav. Aby mohla být použita, muselo dojít k úpravě. Nejrozsáhlejší úpravy se týkaly aktivity hlavní (která se spustí po vstupu do aplikace), ta je nazvaná `CaptureActivity`. Ze zdrojového kódu byla odebrána veškerá možnost nastavení, nastavení knihovny bylo nastaveno „napevno“. Dále byla upravena činnost tlačítek. Byla odebrána možnost vyvolat po stisku tlačítka „menu“ možnosti knihovny. Do aktivity byl vložen kontejner „`ArrayList`“, tento konkrétní kontejner byl zvolen kvůli jeho možnosti odeslat ho zpět do původní aktivity. Do tohoto kontejneru se ukládají výsledky okamžitě po jejich získání. Vzhledem k optimalizaci načítání musel být aktivován nepřetržitý mód („burst mode“), vývojáři knihovny na toto mysleli a připravili tuto možnost do nastavení. Ovšem toto nastavení je ignorováno na zařízeních od firmy Samsung, proto vzhledem k rozdílnému chování ovladačů fotoaparátu a implementaci bylo toto nastavení ignorováno, také byla odebrána možnost načtení jediného kódů. Dále bylo nastaveno tlačítko „zpět“ jako příkaz k ukončení aktivity a k odeslání nazpět do původní aktivity obsahu vytvořeného kontejneru. Při přetváření zdrojových kódů z plně funkční aplikace na knihovnu se vyskytl problém s rozhodováním typu `switch`. Tento druh rozhodování není dovolen, tato restrikce vznikla kvůli optimalizaci rychlosti zpracování knihoven. Z tohoto důvodu musely být příkazy `switch` nahrazeny rozhodováním typu `if-else`.

Vývojové prostředí IntelliJ IDEA při kompilaci aplikace spojí všechny moduly do jedné velké aplikace. Vzhledem k tomu, že knihovna Zxing je samostatně funkční, jak již bylo zmíněno výše, obsahuje všechny složky (mimo složky „assets“) a řídící soubory včetně souboru „Androidmanifest.xml“. Samozřejmě, že v jedné aplikaci se nesmí objevit dva soubory popisující jeho strukturu, proto byla nastavena v ovládání prostředí možnost slučování souborů „Androidmanifest“. Zde evidentně muselo dojít ke konfliktu dat. Z manifestu knihovny byl odebrán parametr nastavující aktivity `CaptureActivity` jako

hlavní a její spuštění po načtení aplikace. Dále byly odebrány nepotřebné aktivity a oprávnění.

Většina oprávnění v aplikaci se týká knihovny Zxing, ale muselo dojít k přidání dalších oprávnění pro lokaci zařízení. Mezi nejdůležitější oprávnění patří:

- CAMERA – oprávnění k přístupu k fotoaparátu,
- VIBRATE – oprávnění pro povolení ovládání vibrací zařízení,
- FLASHLITE – oprávnění pro povolení ovládání přisvětlovací diody,
- ACCESS_FINE_LOCATION – oprávnění zpřístupňující GPS polohu.

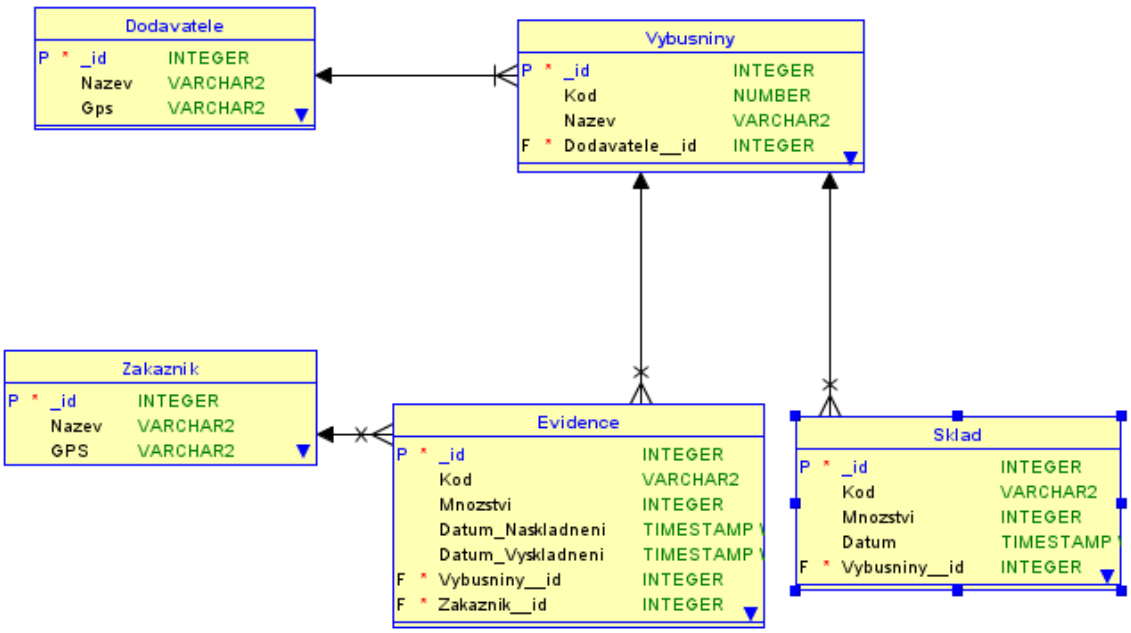
6.2 Návrh tříd

Aplikace byla rozdělena do pěti základních balíčků: `adapter`, `DataObjets`, `fragments`, `hadlers` a základního balíčku `ctecka`. Tyto balíčky slouží k lepší orientaci v souborech tříd. Do základního balíčku `ctecka` jsou centralizovány třídy aktivit, balíček `adapter` slouží k uložení tříd, které jsou určeny k zjednodušení ovládání komplikovaných komponent API Android. V balíčku `DataObjets` se nacházejí třídy sloužící k obalení dat a jejich následné aplikaci. Balíček `handlers` uchovává třídy ovládající moduly (databáze, GPS) nebo externí knihovny (PDF). Poslední balíček obsahuje třídy fragmentů.

Aktivity představují základní třídy, do těchto aktivit jsou vkládány Fragmenty z balíčku `fragments` pomocí příslušného adaptéru. Fragmenty následně pracují s databází pomocí třídy v balíčku `handlers`, případně s dalšími funkcionalitou z balíčku `handlers`.

6.3 Struktura databáze

Vzhledem k rozsahu dat, se kterými aplikace pracuje, bylo nutné vytvořit databázi ve tvaru `SQLite`. Tato databáze obsahuje celkem pět tabulek: Dodavatelé, Výbušniny, Sklad, Sklad, Zákazník, Evidence. ER diagram je nastíněn na Obrázek 5 - ERdiagram. Do tabulky Dodavatelé se ukládají informace o dodavatelích (název a poloha). Atribut poloha slouží ke zrychlené volbě při akci Naskladnění. Tabulka Výbušniny obsahuje informace o jednotlivých produktech, se kterými zákazník obchoduje. Tabulka Sklad sdružuje informace o naskladněných položkách, a to: kód naskladněné položky, id dodavatele, datum a čas naskladnění, informaci o naskladněném produktu. Podobně vyhlížející tabulka je Evidence, ta navíc obsahuje informace o vyskladnění produktu (u kterého zákazníka s datovou známkou). Jako poslední se v modelu vyskytuje tabulka Zákazník. V této tabulce se nachází informace o zákaznících firmy, ke kterým se dodávají výbušniny. Jedná se o informace o jméně zákazníka a poloze vyskladnění.



Obrázek 5 - ERdiagram

Protože v provozu aplikace může nastat situace, kdy dojde k rozšíření působení firmy o nové dodavatele i zákazníky, byla do aplikace doplněna možnost editovat databázi. Při nechtěném poškození databáze nebo při prvotní instalaci aplikace je program vybaven základními daty v XML souborech. Samotné SQL příkazy pro vytvoření tabulek se nacházejí ve složce `asset`. Pro každou tabulku je určen jeden soubor se značkami pojmenovanými podle atributů v tabulce.

Ukázka zálohovacího souboru pro Dodavatele:

```
<database>
    <dodavatel>
        <nazev>Explosia a.s.</nazev>
        <gps>50°3'41.289"N, 15°43'40.909"E</gps>
    </dodavatel
</database>
```

6.4 Návrh Android standardů

Vzhledem k tomu, že aplikace je určená pro Android od verze 4, jsou použity návrhové techniky doporučované společností Google. Jedná se především implementaci Fragmentů a ActionBar menu.

Použití Fragmentů výrazně zmenšilo celkový počet aktivit, a tím pádem také manifestu. Nastavení aktivit v manifestu se tedy velmi zjednoduší a zmenší se duplicita nastavení. Fragmenty se ale musí vkládat do obalovací třídy *`TabsPagerAdapter`*, která je potomkem třídy z knihovny android.support.v4 *`FragmentPagerAdapter`*. Tato třída bohužel neuchovává instance o Fragmentech, což zapříčiní problémy při implementaci.

6.4.1 Activity

Celá aplikace je tvořena ze čtyř základních aktivit: `StartUp2`, `DatabaseFragmentActitivy`, `ModifyDatabase`, `SearchResultActivity`. Jako hlavní aktivita slouží StartUp2, která v sobě ukrývá čtyři Fragmenty. Aktivita `DatabaseFragmentActitivy` slouží pro zobrazení a ovládání databáze. `ModifyDatabase` poslouží pro uložení nového záznamu do databáze a také pro úpravy stávajících dat v databázi. Poslední aktivita slouží pro zobrazení výsledku vyhledávání.

Každá aktivita má vlastní vzhled, který je uložen v souboru formátu XML. Tyto soubory jsou uloženy ve složce `res/layout` a jsou pojmenovány shodně s názvem aktivity. V aktivitách určených pro držení Fragmentů se nachází jediná komponenta `ViewPager`. Do této komponenty se následně dynamicky vkládají Fragmenty. V rozložení pro zobrazení výsledku vyhledávání se nachází pouze jedno textové pole pro zobrazení výsledku. Až v posledním rozložení se nachází větší množství prvků. Ty jsou dynamicky měněny podle potřeby zobrazení. Základním prvkem, který drží rozložení stránky, je `TableLayout`, do tohoto jsou vkládány řádky a do těchto řádků jsou vkládány textové prvky pro popis a prvky, které umožňují vstup pro uživatele.

Aktivita `StartUp2`, i přes fakt, že je nejpoužívanější z celé aplikace, je velmi jednoduchá. Složitější chování je přenecháno až fragmentům. Aktivita implementuje pouze ActionBar menu a `TabsPagerAdapter`. Podobně je to i s aktivitou `DatabaseFragmentActitivy`, problém s instancemi fragmentů bude popsán v dalších kapitolách.

Nejsložitější aktivitou je `ModifyDatabase`, tato jediná aktivita zastává funkci dvou aktivit, tedy přidání a editace. Umožnuje uživateli přidávat nový záznam do databáze a umožňuje modifikovat stávající data v databázi. Na počátku životního cyklu aktivity se získají informace přiložené aktivitou, která ji vytvořila. Informace jsou uloženy ve třídě `Intent`, jedná se hlavně o informaci „update“. Tato hodnota typu `boolean` určuje, jestli data vkládáme, nebo je upravujeme. Další důležitou informací je řetězec „`dtbName`“. Tento řetězec určuje, s jakou tabulkou se bude pracovat. Následuje rozhodnutí o tom, které komponenty se mají zobrazit a které ne. Pokud se jedná o tabulky „`Zakaznik`“ nebo „`Dodavatele`“, vkládají se pouze komponenty pro úpravu názvu a GPS polohy. Pokud se jedná o tabulku „`Vybusniny`“, jsou vloženy komponenty: název, kód a komponenta „`Spinner`“. Tato komponenta je naplněna názvy výrobců z databáze. Na začátku plnění je dotaz odeslán do databáze, o ten se stará metoda `getDodavatele` z třídy `DatabaseHelper`, třída je vytvořena na začátku metody `onCreate`. Metoda vrací instanci třídy `Cursor`, která je naplněna názvy Dodavatelů. Pro naplnění `Spinneru` musí být názvy v kontejner typu `ArrayList`. Převod zajistíme jednoduchým projitím záznamu:

```
while (c.moveToNext()) {
    list.add(c.getString(0));
            }
```

Přiřazení dat z kontejneru do `spinneru` je implementováno pomocí třídy `ArrayAdapter`. Následovně:

```
   ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item, list);
   dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dr
opdown_item);
   spinner.setAdapter(dataAdapter);
```

Po dokončení zadávání údajů a stisknutí tlačítka potvrzení dochází k rozhodnutí mezi uložením nových informací (metoda `addToDTB`) nebo úpravou záznamu (metoda `updateDTB`). Pokud se jedná o vložení nového záznamu, jsou získána data z příslušných komponent a následně volány metody ze třídy `DatabaseHelper` pro správnou tabulku. Pokud se jedná o úpravu záznamů v databázi, jsou stejným způsobem získána data a jsou volány metody pro úpravu dat v databázi, následně je aktivita ukončena metodou `finish`.

Pokud se jedná manipulaci s tabulkami „`Zakaznik`“ nebo „`Dodavatele`“, pole GPS je kontrolováno pomocí metody `checkGps`. Tato metoda má jeden vstupní parametr, kterým je hodnota pole GPS. Kontrola formátu dat probíhá pomocí regulárního výrazu, data musí ve formátu „00°00’00.000”N, 00°00’00.000”E“.

6.4.2 Action bar menu

ActionBar menu se dá poměrně snadno vložit do zdrojového kódu aktivity pomocí metody `onCreateOptionsMenu`.

Příklad vložení menu:

```
   @Override
       public boolean onCreateOptionsMenu(Menu menu){
           MenuInflater inflater=getMenuInflater();
           inflater.inflate(R.menu.database_menu,menu);
           return super.onCreateOptionsMenu(menu);
       }
```

Samostatný vzhled menu lišty se připravuje do souboru XML ve složce res/menu. Do tohoto souboru se vkládají značky `group` nebo `item`, případně se dá spojit `group` a do této značky se dá vložit další značky `item`.

V aplikaci je použita zpětná navigace integrovaná do horního panelu menu. Tuto možnost lze jednoduše aktivovat v manifestu u jednotlivých aktivit a nastavení hodnoty `android:parentActivityName`.

Vlastní zachytávání a reagování na akce, které se dějí v Actionbar menu, se ošetřuje v metodě `onOptionsItemSelected`. Pokud tuto metodu ovšem přepíšeme, přestává být funkční zpětná navigace a je třeba reagovat i na horní tlačítko zpět. Jako vstupní parametr je třída `MenuItem`. Z instance této třídy můžeme pomocí metody `getItemId` zjistit, které tlačítko bylo zvoleno. Pro zachování zpětné navigace musí aplikace reagovat na hodnotu `android.R.id.home`, a to tak, že vyvolá statickou třídu `NavUtils` a její metodu `navigateUpFromSameTask` a jako parametr dodá instanci aktivity `this`.

Pro ovládání aplikace byly vloženy ikony přímo od společnosti Google. Jedná se o tlačítka: potvrzení, zamítnutí, editace, přidání, rozbalení nabídky a vyhledání. Všechny ikony jsou používány podle návrhového vzoru a doporučení vývojářů operačního systému Android. Tyto ikony se nacházejí v adresáři `res/drawable`, případně dalších adresářích, do kterých program přistupuje podle rozlišení daného telefonu. Jelikož aplikace podporuje rozlišení hdpi, ldpi, mdpi, xhdpi a xxhdpi, tak byly dodány ikony s patřičným rozlišením.

Speciálním chováním se vyčleňuje akce hledání kódů s ikonou `search`. Tuto funkci lze použít pouze na některých Fragmentech, proto musí být na Fragmentech, kde není možné použití této ikony, použito nezobrazit. Tato funkcionalita je řešena ve třídě `StartUp2` a stará se o ni přepsaná metoda `onTabSelected`. Před samostatným rozhodnutím musíme ověřit existenci instance menu (což je globální proměnná typu `Menu`). Toto je zřejmé z životního cyklu aktivity, kdy se před vytvořením ActionBar menu vytváří Fragmenty, nastavuje se první Fragment a po nastavení prvního Fragmentu se spouští metoda `onTabSelected`. Po zjištění existence lze rozhodnout, zda má být zobrazena ikona.

Ukázka rozhodování:

```
if(menu!=null) {
   if (tab.getPosition() < 2) {
       menu.findItem(R.id.action_search).setVisible(false);
   } else {
       menu.findItem(R.id.action_search).setVisible(true);
```

Pro chod vyhledávání podle standardu musí být při vytváření menu v metodě `onCreateOptionsMenu` vytvořena třída `SearchView` z již existujícího menu a následně nastavení možnosti vyhledávání:

```
searchView.setSearchableInfo(searchManager.getSearchableInfo(
getComponentName()));
```

Po dokončení zadání textu uživatele a kliknutí na tlačítko vyhledat se odešle obsah v textovém poli do nové aktivity. O kterou aktivitu se jedná, to určuje atribut v souboru manifest. Tento atribut je zapsán u aktivity „zdrojové“. Jedná se o tag:

```
<meta-data android:name="android.app.default_searchable"
android:value=".SearchResultsActivity" />.
```

Cílovou aktivitu je potřeba opatřit konkrétním *intent*-fitrem, a to: `<action android:name="android.intent.action.SEARCH" />`. Tento filtr zajistí, že po vzniku aktivity (*SearchResultsActivity*) se okamžitě přejde do metody `handleIntent`, kde již lze zpracovat příchozí informace, vyhledat je v databázi a případně je i exportovat.

U menu tlačítek v aktivitě `DtabaseFragmentActitivy` se převážně spouští nová aktivita se vstupními parametry odeslanými pomocí třídy `Intent`. Podrobněji budou tyto akce popsány v další kapitole s problémem Fragmentů.

6.4.3 Fragment tab manager

Pro zobrazení jedné obrazovky jako jednoho Fragmentu je zapotřebí použít třídu `FragmentPagerAdapter`, která se nachází v externí knihovně android.support.v4. Pro zjednodušení použití byla vytvořena nová třída `TabsPagerAdapter`, která je potomkem třídy `FragmentPagerAdapter`. Tato třída zajišťuje chod Fragmentů na úvodní obrazovce, tedy Fragmenty: Naskladnění, Vyskladnění, Sklad, Evidence. V této třídě jsou přepsány metody:

- `getItem` – metoda po zadání pořadového čísla navrátí instanci požadovaného Fragmentu,
- `getCount` – metoda vrací počet Fragmentů, které třída obsahuje.

Další třída stejného typu je `TabsPagerAdapterDatabase`. Tato třída kromě obsahu třídy `FragmentPagerAdapter` obsahuje kontejner s uloženými Fragmenty a metodu pro navrácení všech Fragmentů uložených v kontejneru. Toto rozšíření se týká nedostatku třídy `FragmentPagerAdapter`, který neumožňuje navrácení již vytvořených Fragmentů.

Tyto adaptéry byly separovány do zvláštního balíčku `adapter`, což zajišťuje rychlejší orientaci v kódu. V tomto balíčku je obsažena ještě jedna třída `ExpandableListAdapter`. Tento adaptér slouží ke snazší páci s rozvinovacími seznamy u obrazovek Evidence a Sklad. Třída je potomek základní třídy `BaseExpandableListAdapter` a její hlavní funkcí je zjednodušení práce s touto komponentou. Mezi hlavní rozšíření, oproti třídě rodiče, patří uchovávání dat v nadpisech buněk. Důležité metody `getGroupView` a `getChildView` navracejí třídu `View`, která se dále v programu používá pro práci s obsahem seznamu.

Ukázka třídy:

```
@Override
public View getChildView(int groupPosition, final int childPosition,
boolean isLastChild, View convertView, ViewGroup parent) {
     final String childText = (String) getChild(groupPosition,
childPosition);
     if (convertView == null) {
        LayoutInflater infalInflater = (LayoutInflater)
this._context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        convertView = infalInflater.inflate(R.layout.list_item, null);
           }
     TextView txtListChild=(TextView)
convertView.findViewById(R.id.lblListItem);
     txtListChild.setText(childText);
     return convertView;
       }
```

6.4.4 Fragment

Všechny třídy typu Fragment jsou centralizovány do balíčku `fragments`. Tyto třídy můžeme rozdělit do dvou základních skupin: Fragmenty pro základní akce a Fragmenty pro zobrazení databáze. Speciální Fragment je třída `DatabaseFragment`. Tato třída je potomek třídy `Fragment`, ale je třídou abstraktní. Třídu `Fragment` rozšiřuje o řetězec `selectedRow` a metody pro nastavení a získání tohoto řetězce. U této třídy jsou dále implementovány třídy pro zobrazení dat z databáze.

Ke každému Fragmentu musí existovat popisovací soubor XML, který popisuje rozložení komponent na obrazovce. Tyto soubory se nacházejí ve složce `res/layout` a jsou pojmenovány podle vzoru „fragment_nazevfragmentu“. Popisovací soubory Fragmentů pro zobrazení dat z databáze obsahují pouze prázdnou tabulku (`TableLayout`). Ostatní obsahují většinu komponent, které se zobrazí uživateli.

Příklad schématu pro vyskladnění zboží:

```
   <TextView android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:textAppearance="?android:attr/textAppearanceLarge"
           android:text="Vyskladnění"
           android:id="@+id/textView" android:layout_gravity
="center_horizontal"/>
   <Spinner android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/spinner"/>
   <Button android:layout_width="match_parent"
           android:layout_height="wrap_content"
           android:text="Snimej"
           android:id="@+id/buttonSnimej"
           android:layout_gravity="center_horizontal"/>
```

Fragmenty neuchovávají ani nemohou uchovávat `Contex`. Tato třída obsahuje informace a metody pro ovládání aplikace. Proto třídy typu `Fragment` obsahují metodu pro získání aktivity, ve které se nachází Fragment (`getActivity`).

6.5 Fáze aplikace

6.5.1 Naskladnění

Při začátku životního cyklu Fragmentu `NaskladneniFragment` je volána metoda `onCreateView`. V této metodě se připravují kompetenty pro zobrazení na displeji zařízení. Toto se připravuje do třídy `View` a proměnné `rootView`. Následně připraví třída `GPSTracer` do proměnné `gps`, poté následuje rozhodnutí o stavu GPS modulu. Ukázka:

```
   if(gps.canGetLocation()){
           double latitude = gps.getLatitude();
           double longitude = gps.getLongitude();
      }else{
           gps.showSettingsAlert();}
```

Pokud je GPS modul zakázán a je zakázaná i lokace pomocí mobilních sítí, je zobrazen Dialog s rychlou volbou, která odkáže uživatele do nastavení zařízení na volbu lokace. Následuje načtení dat z databáze (data o dodavatelích a jejich výrobcích), protože jsou tato data následně uložena jako pouhý řetězec do výběrového prvku, do kterého nelze uložit další hodnotu `id` pro další práci. Byla vytvořena třída `StringWithTag`, do této třídy jsou vkládány hodnoty názvu a údaj `id`.

Samotná třída vypadá následovně:

```
public String string;
public String tag;

public StringWithTag(String stringPart, String tagPart) {
    string = stringPart;
    tag = tagPart;
}
@Override
public String toString() {
    return string;
}
```

Metoda `toString` vrací hodnotu řetězce nazpět, a tím pádem zachovává funkčnost třídy `String`, navíc třída obsahuje metody pro nastavování a získávání hodnot z proměnné `tag`.

Po získání dat z tabulky dodavatelů dojde k naplnění kontejneru názvy dodavatelů a přidělení čísla *id*. Pokud je aktivní GPS modul, určí se zároveň nejbližší dodavatel porovnáním polohy dodavatele s aktuální vzdáleností (hledání minima). K výpočtu vzdálenosti se používá metoda třídy `GPSTracker getDistastance`, do této třídy přichází jako vstupní parametr záznam z databáze. Metoda převede vstupní řetězec na jednotlivé složky (stupně, minuty a sekundy) a převede do stupnicového tvaru. Následně se použije metoda `distanceTo`, která navrátí vzdálenost v metrech.

Na konci inicializační metody se vytvářejí Listnery pro tlačítko začátku snímání kódů a změna výběru dodavatele. Listener na výběru naslouchá vybrání jednoho z dodavatelů. Pokud k této události dojde, odešle se identifikační číslo dodavatele do metody `fillSpiner2`. Tato metoda slouží k nahrání obsahu do druhého výběru. Podle vstupního parametru (identifikační číslo dodavatele) se naleznou odpovídající záznamy v databázi výbušnin (metoda `getVybusninu`) a následně se uloží informace o výbušninách do objektů třídy `StringWithTag` (identifikační číslo, název výbušniny) a tento objekt do kontejneru. Tento kontejner je následně přiřazen do výběrové komponenty pomocí třídy `ArrayAdapter`. Reakci Listeneru na tlačítko je zapotřebí generovat v kódu (ne v XML šabloně vzhledu, jak je zvykem), kvůli potlačení této funkcionality v zobrazení pomocí Fragmentů. Před přejitím do jiné aktivity je zapotřebí uložit nastavení výběrů. O uložení stavu Fragmentu se stará metoda `onSaveInstanceState`. Pro vlastní přejití do aktivity snímání se použije třída `Intent`:

```
Intent intent=new Intent(getActivity(), CaptureActivity.class);
startActivityForResult(intent,0);
```

Po návratu do Fragmentu je volána metoda `onActivityResult`. V této metodě proběhne získání kódů ze třídy `Bundle`, data jsou uložena v kontejneru typu `ArrayList`. Tento kontejner umožnuje ukládání duplicitních kódů a při načítání je běžné, že knihovna `Zxing` stihne nasnímat kód vícekrát za sebou, než stihne uživatel přesměrovat svoji pozornost nebo ukončit zadávání. Proto musejí být tyto duplicity odstraněny, toho je dosaženo převedením kolekce do kontejneru typu `HashSet`. Následně jsou data předána metodě `saveReadedData`, která vytvoří nové datové objekty `Polozka`. Do těchto objektů jsou uložena data o kódu a identifikační číslo výbušniny, která byla snímána. V další části metody jsou všechny nově vytvořené položky testovány, zda se v databázi nevyskytují, pokud k takové skutečnosti dojde, je položka okamžitě odebrána ze seznamu. Na konci metody se data odešlou do třídy starající se o databázi pomocí metody `insertSklad`.

Ukázka procházení:

```
for(Polozka item : items){
    if(myDbHelper.isInDatabase(item.getCode())){
       items.remove(item);
       }
}
```

6.5.2 Vyskladnění

Fragment Vyskladnění obsahuje velmi podobný kód jako předchozí obrazovka Naskladnění. Rozdíly však lze nalézt v rozložení komponent. Na této obrazovce se nachází pouze jeden výběr, který reprezentuje Zákazníky firmy. Tato výběrová komponenta je plněna z databáze za použití třídy `StringWithTag`. Fragment je na začátku jeho života odkázán na metodu `onCreateView`, ve které se nachází kód pro výběr nejbližšího zákazníka, stejně jako u Fragmentu Naskladnění. A dále Listener pro tlačítko (znovu totožný s kódem zmíněným v předešlé kapitole).

Rozdíl je v chování po navrácení kódu ze skeneru. Zde se odkazuje aplikace na metodu `saveReadedData`. V této metodě se porovnají naskenované kódy s kódy uloženími v tabulce Sklad, a to následujícím způsobem. Do metody přichází jako vstupní parametr `HashSet` obsahující naskenovaná data. Na začátku metody se získají všechny naskladněné kódy (pomocí metody `getPolozkyFromSklad` z třídy `DataBaseHelper`) a uloží se do kontejneru typu `ArrayList<Polozka>` (`Polozka` datová třída obsahující data k danému skenu). Následně je pomocí iterátoru projen seznam nových kódu. U každé položky je volána metoda `isInStorage`, ve které se znovu pomocí iterátoru prozkoumá existence kódu ve stažené tabulce. Pokud je kód nalezen, je navrácena třída `Polozka` obsahující všechna potřebná data, pokud ke shodě nedojde, návratová hodnota je rovna `null`. Zpět v metodě `saveReadedData` se porovná výsledek vyhledávání, pokud se navrátila prázdná data, je s touto skutečností obeznámen uživatel a pokračuje se dalším kódem. Pokud navracený objekt není roven `null`, přiřadí se k položce číslo zákazníka a tato položka se uloží do výsledkového kontejneru `items`. Po dokončení procházení se tato data přesunou z tabulky sklad do tabulky evidence (popsáno v následující kapitole).

Ukázka metody *saveReadedData*:

```
   ArrayList<Polozka> storeItems=myDbHelper.getPolozkyFromSklad();
   ArrayList<Polozka> items=new ArrayList<Polozka>();
   Iterator<String> it=hset.iterator();
   while(it.hasNext()){
         Polozka item=isInStorage(storeItems,it.next());
         if(item!=null){
            item.setZakaznik(Integer.toString(spinnerSelected));
            items.add(item);
         } else{
               Toast.makeText(getActivity().getApplicationContext(),
"Kód se nenachází na skladě. Nelze převést do evidence.",
Toast.LENGTH_LONG).show();
         }
   }
   myDbHelper.moveToEvidence(items);
```

6.5.3 Sklad

Stránka sklad je tvořena rozjížděcím seznamem dat, tuto komponentu tvoří třída `ExpandableListView` se stejnojmennou komponentou v popisovacím souboru `fragment_sklad.xml`. Pro snazší ovládání je použita předpřipravená ovládací metoda v balíčku `adapter`, třída `ExpandableListAdapter`. V kódu je držena instance v globální proměnné `listAdapter`.

Při přípravě zobrazení dat se volá metoda `prepareListData`. Tato metoda má za úkol připravit data pro seznam. Na začátku se zavolá metoda `getVybusninyFromSklad`, tato metoda se nachází ve třídě `DataBaseHelper`, a připraví kompletní data na použití (pokud existují záznamy v databázi). Následuje kontrola návratové hodnoty. Pokud se navrátila prázdná kolekce (hodnota `null`), je do seznamu vypsána informace o neexistenci dat.

Na rozjížděcí seznam je přidán Listener kontrolující dlouhý stisk položky v seznamu (`setOnItemLongClickListener`). V této třídě je potřeba přepsat metodu `onItemLongClick`. Do této metody je dopsán kód pro vybudování dialogového okna, pokládající otázku o vytvoření PDF souboru na paměťovou kartu telefonu.

6.5.4 Evidence

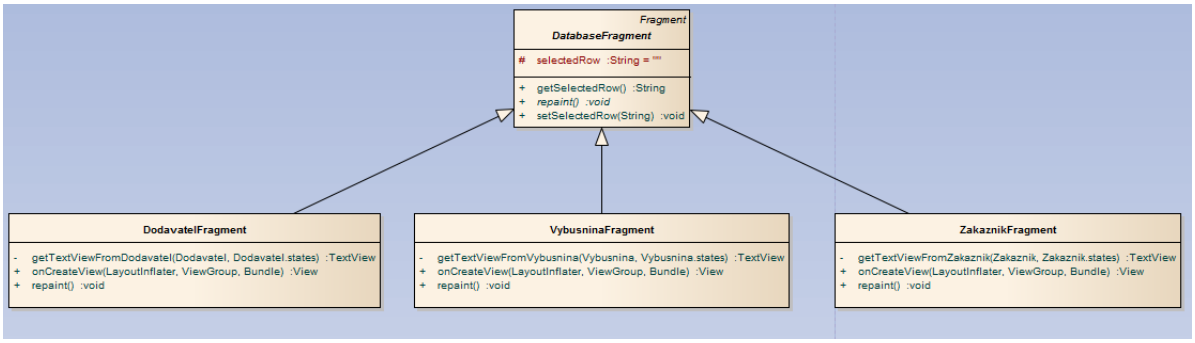
Fragment Evidence se prakticky neliší od Fragmentu Sklad. Jediným rozdílem je použití jiné metody pro získání dat z databáze, jedná se o metodu `getVybusninyFromEvidence`.

6.5.5 Databáze

Aktivita `DatabaseFragmentActitivy` obsahuje hlavní komponentu držící tři fragmenty ve třídě `TabsPagerAdapterDatabase`. Všechny tři Fragmenty jsou si navzájem velmi podobné. Jejich hlavní rozdíl je v práci s rozlišnými tabulkami. Při implementaci

zvýrazňování řádku nastal problém s předáváním instancí mezi aktivitou a adaptérem, který spravuje Fragmenty. Třída `FragmentPagerAdapter` nedokáže navrátit instanci právě zobrazovaného Fragmentu. Protože jsou ovládací tlačítka Actionbar menu vázána na aktivitu Fragment, musel být implementován způsob komunikace mezi těmito dvěma třídami.

Z Fragmentu je třeba zjistit, jaký řádek tabulky je vybrán a také spustit překreslení dat. Proto byla vytvořena abstraktní třída `DatabaseFragment`, která obsahuje řetězec obsahující informaci o tom, který řádek byl vybrán, a abstraktní metoda `repaint`. Všechny další Fragmenty pracující s daty z databáze jsou potomci této třídy, tedy musí implementovat metodu `repaint` a mohou pracovat s proměnou `selectedRow` obsahující identifikaci vybraného řádku (zkrácený UML diagram zobrazen na Obrázek 6 - Třídy Fragmentů). Ve třídě `FragmentPagerAdapter` musí existovat kolekce (`ArrayList`), která bude obsahovat instance používaných Fragmentů, tedy třídy typu `DatabaseFragment`.



Obrázek 6 - Třídy Fragmentů

V jednotlivých Fragmentech se vypisují data do komponenty `TableLayou`, a to postupně podle řádků. Na začátku výpisu je načtení všech z dané tabulky, metody jsou pojmenovány podle tabulky např. `getAllDodavatel`. Z metody se vrátí naplněná kolekce `ArrayList`, která obsahuje datové objekty typu `Dodavetel`, tato třída pouze organizuje data pro snazší práci. Třída obsahuje pouze metody pro nastavení a získání hodnot z proměnných. Po získání všech dat se cyklicky vytvoří jednotlivé řádky. Do těchto řádků se přidají komponenty pro text pomocí metody `getTextViewFromDodavatel`. Tato metoda vytvoří instanci třídy `TextView` podle typu tabulky a právě požadovaných dat. Ke každému řádku se přidá Listener, který naslouchá na stisk řádku. Ukázka třídy viz. Příloha C.

Pro vybarvení řádku se použije předcházející kód. V této ukázce se kontrolují tři stavy, a to: řádek nebyl označen, řádek byl označen a je označený jiný řádek. Tyto stavy se porovnávají pomocí proměnné `selectedRow`. Pokud tato proměnná je shodná s proměnnou *id*, což je hodnota identifikačního čísla řádku, na který bylo kliknuto, tak to znamená, že si přejeme odznačit řádek, tedy určí se, zda řádek sudý nebo lichý, a podle výsledku se obarví barvou světlejší nebo tmavší. Pokud k tomuto stavu nedošlo

a proměnná `selectedRow` není prázdná, znamená tato situace, že již jeden řádek je označen a nyní se musí označit jiný. Nejprve nalezne instanci řádku pomocí metody `findViewById` a následně ji obarví podle toho, zda se jedná o lichý nebo sudý řádek. A následně se obarví požadovaný řádek na oranžovo a uloží se hodnota *id* do proměnné `selectedRow`. Poslední možná situace je, že ještě nebyl žádný řádek. V této situaci postačí obarvit řádek na barvu oranžová a uložit hodnotu *id* do proměnné `selectedRow`.

Třída musí implementovat metodu `repaint`, tato metoda je prakticky totožná s předešlou popisovanou metodou `onCreateView`. Tuto metodu nelze použít přímo při vytváření komponent kvůli jejich neexistenci. Tato třída je následně volána při návratu do aktivity a poslouží k aktualizaci obsahu tabulky.

Pokud uživatel stiskne tlačítko Actionbar menu, musí se implementace přenechat aktivitě `DatabaseFragmentActitivy`, a to konkrétně metodě `onOptionsItemSelected`. Do metody jako vstupní parametr přichází položka, která byla vybrána a následně je rozhodnuto o tom, která položka byla aktivována a jaká se má provést akce. Toto zajišťuje příkaz `switch`. Rozhoduje se o následujících tlačítkách:

- Tlačítko zpět – návrat do předchozí aktivity (`StartUp2`).
- Přidání nového prvku do databáze – rozhodne se, do jaké tabulku chce uživatel přidat obsah s pomocí proměnné `selectedTab` (ta je měněna při přechodu do jiného Fragmentu) a následně se připraví start nové aktivity `ModifyDatabase`. Do startu se přidají údaje o tabulce a informace, že se nejedná o úpravu dat.
- Úprava existujícího prvku v tabulce – zde se stejným způsobem rozhodne o tabulce. A připraví se start aktivity `ModifyDatabase`. Ovšem do informací se přidá hodnota `id`, která reprezentuje identifikační číslo vybraného záznamu a také informace o tom, že se jedná o úpravu tabulky.
- Smazání záznamu z tabulky – pokud je zvolen záznam, je před uživatele položena otázka, zda opravdu chce smazat trvale záznam z databáze. Pokud odpoví kladně, metoda `actuallyDelete` spustí patřičnou metodu ze třídy `DataBaseHelper`
- Nahrání zálohy databáze – zavolá se metoda `reload` ze třídy `DataBaseHelper`. Tato metoda přepíše obsah databáze do stavu po nainstalování aplikace.

6.5.6 Ovladač databáze

Obsluha databáze je centralizována do třídy `DataBaseHelper` v balíčku `handlers`. Tato třída obsahuje veškeré metody pracující s databází. Pro bezproblémový chod aplikace obsahuje třída metody zajišťující nahrání prvotních dat ze zálohovaných souborů XML.

Třída `DataBaseHelper` musí obsahovat metody pro prvotní vytvoření databáze a také metodu pro povýšení databáze na novější verzi. Při prvním vytváření objektu `DataBaseHelper` se spustí metoda `onCreate`, tato metoda se postará o vytvoření tabulek a následné naplnění daty z XML souborů. Předpis SQL příkazů pro vytvoření tabulek se nachází v souboru string.xml. Data určená pro vložení do databáze se nachází v dalších XML souborech nacházející se ve složce `assets` (soubory byly popsány v předchozích kapitolách).

Data uložená v souborech se čtou pomocí dvou metod `readXmlDatabase` a `parseXML`. O rozkódování dat se stará třída `XmlPullParser`, tato třída vyžaduje ke čtení třídu `InputSream` odkazující na soubor XML, jelikož jsou sobory uloženy ve složce `assets` získání `InputStream` instance zabezpečí třída z balíčku Android Api `AssetManager`.

Vlastní data se parsují v další metodě `parseXML`. V této metodě se postupně čtou tagy XML souboru a podle značky se zapisuje jejich obsah do kolekce typu `ArrayList`. Ukázku kódu pracující s data viz. Příloha D.

6.5.7 Export do PDF

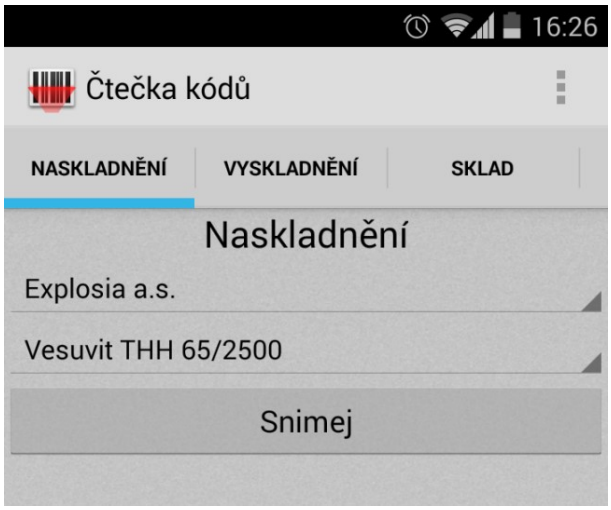
O export do souboru formátu PDF se stará třída z balíčku `handlers exportPDF`. Pro vlastní export stačí pouze vytvořit instanci třídy a po vytvoření instance se automaticky vytvoří a uloží soubor ve formátu PDF do složky uživatelem přístupné (`download`).

Konstruktor třídy vyžaduje vstupní parametr formátovaný text, který se má vytisknout do souboru. O tištění se stará externí knihovna ITEXT. Třída obsahuje čtyři základní metody. První metoda připraví soubor a cestu k uložení, následně předá instanci souboru další třídě `addMetaData`. Tato třída vloží do souboru data o názvu a tvůrci dokumentu. Další metoda, které je předána instanci, je metoda `addtitlePage`. V této metodě se tiskne vlastní obsah do souboru. Postupně se vloží jednotlivé paragrafy. Mezi paragrafy se vkládají mezery, pro zjednodušení kódu je vkládání volných řádků izolováno do metody `addEmptyLine`. Po dokončení vkládání obsahu se dokument uzavírá metodou `close`. Po tomto uzavření dojde k uložení souboru na paměť SD.

7 Ovládání aplikace

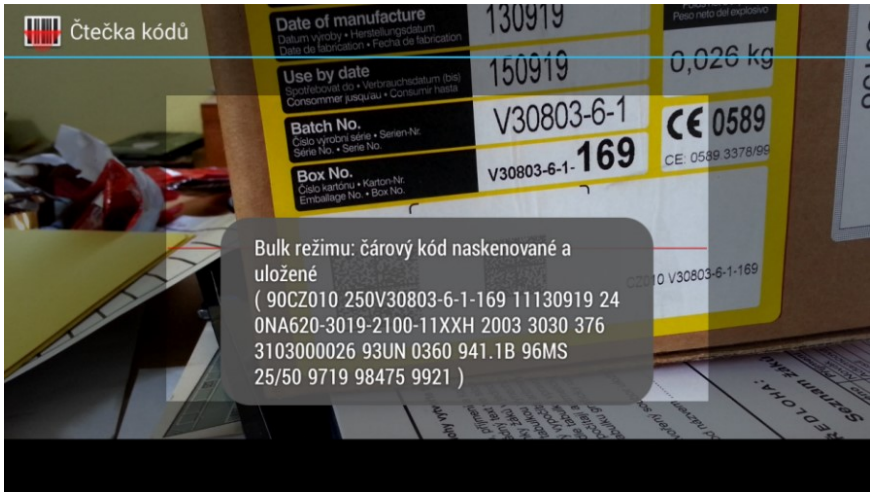
Aplikace je rozvržena podle standardů operačního systému Android verze 4 do Fragmentů a ovládacích prvků v ActionBar menu.

Po spuštění aplikace je zobrazena obrazovka, na které si lze povšimnout v horní části displeje ikony aplikace a také název aplikace. V pravém horním rohu lišty se nachází tlačítko, pod kterým jsou rozšířené možnosti. V této nabídce se nachází možnost přechodu do přehledu databáze. Pod touto lištou je umístěn výpis záložek (viz Obrázek 7). Tyto záložky představují jednotlivé Fragmenty. Jedná se o položky: Naskladnění, Vyskladnění, Sklad, Evidence. Mezi těmito Fragmenty lze přepínat stiskem názvu Fragmentu nebo přejetím prstem přes obrazovku doleva nebo doprava.



Obrázek 7- Fragment Naskladnění

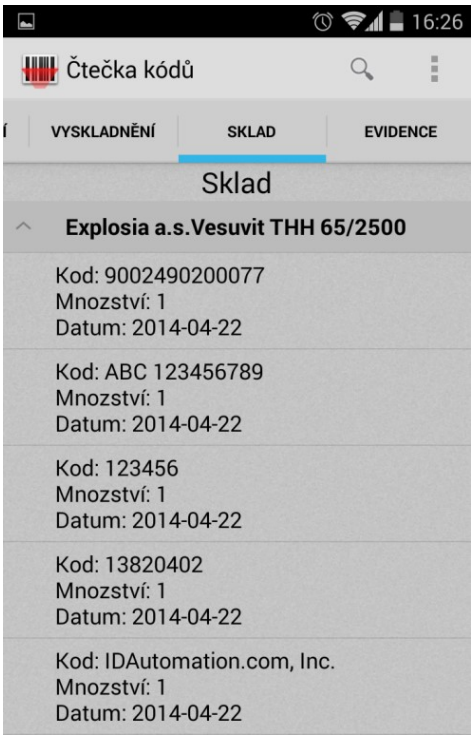
Na první obrazovce se nachází Fragment Naskladnění. Zde je možnost naskladnit nové zboží do skladu. Na displeji jsou zobrazeny tři položky, a to: výběr dodavatelské firmy, výběr ze zboží aktuálně vybraného dodavatele a tlačítko pro začátek snímání. Fragment Vyskladnění obsahuje výběr zákazníka, u kterého prodáváme zboží, a tlačítko na snímání prvků. U obou těchto Fragmentů se program snaží vybrat dodavatele/zákazníka, ke kterému je zařízení nejvíce přiblížené. Této funkce je dosaženo pomocí GPS modulu.



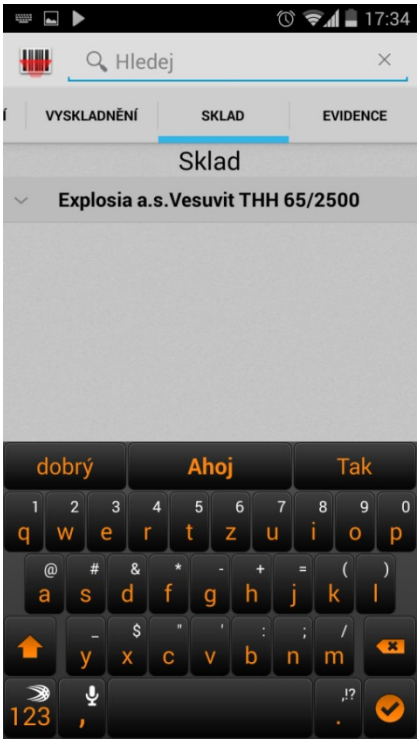
Obrázek 8- Snímání

Po stisku tlačítka Snímej se aplikace přepne do snímacího režimu. V tomto režimu pomocí fotoaparátu jsou hledány kódy, které by případně aplikace dokázala dekódovat. Načtení kódu probíhá zcela automaticky a od uživatele se žádá pouze zaměření telefonu na daný kód. Po úspěšném získání kódu aplikace vypíše krátkou hlášku a načtený kód (viz Obrázek 8). Následně pokračuje ve snímání. V případě nízkého osvětlení lze aktivovat přisvětlovací diodu (pokud je jí zařízení vybaveno), a to stisknutím tlačítka pro zesílení hlasitosti, dioda se dá vypnout tlačítkem zeslabit. Po dokončení načítání všech kódů a návrat do nabídky je zapotřebí stisknout tlačítko zpět.

Fragment Sklad obsahuje firmou nakoupený materiál, který doposud neprodala. Hlavním obsahem tohoto zobrazení je skladový výpis (viz Obrázek 9). Tento výpis je řazen do položek ve tvaru Název dodavatelské firmy mezera Název produktu. V levém rohu každé položky se nachází šipka, kterou lze rozbalit všechny záznamy pro aktuálně vybraný produkt. Poslední obrazovka obsahuje Evidenci prodaných výrobků. Zde jsou položky seskupovány podle zákazníka, kterému byly doručeny. Pokud bude požadován export jedné položky, je možnost dlouhé stisku položky a tato položka bude následně exportována do dokumentu PDF s patřičnými údaji. Na těchto obrazovkách přibylo tlačítko pro vyhledávání kódů. Po stisknutí tohoto tlačítka se horní lišta promění na zadávací pole, do kterého lze zapsat kód, jaký si uživatel přeje nalézt. Také se automaticky aktivuje systémová klávesnice. Vyhledávání je zobrazeno na Obrázek 10. Po dokončení zadání vyhledávacího výrazu a stisku tlačítka potvrdit přejde aplikace na novu obrazovku s výsledky vyhledávání.

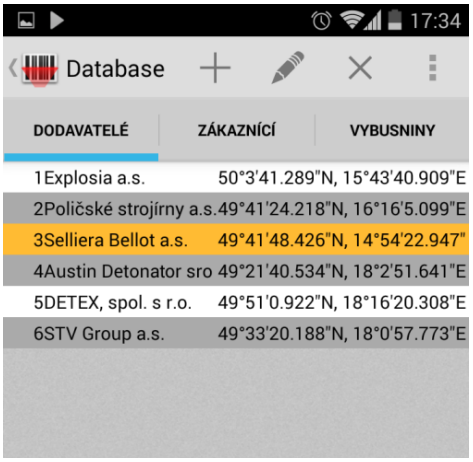


Obrázek 9 - Sklad



Obrázek 10 - Vyhledávání

Po přechodu na obrazovku pro správu databáze si lze povšimnout změny horního panelu ActionBar. Zde přibyly zleva následující položky: navigace zpět na původní aktivitu, přidání nové položky do aktuální tabulky, editace aktuálně vybrané položky, smazání aktuálně vybrané položky a rozšířené možnosti, ve kterých se nachází možnost obnovit databázi do stavu po nainstalování aplikace. V liště pod ActionBarem se nachází stejná Navigace jako v předchozí obrazovce. Pod ní se nachází již výpis všech položek v aktuálně vybrané tabulce. Pro výběr jedné položky (řádku) stačí stisk zamýšleného řádku. Řádek okamžitě změní barvu na oranžovou (viz Obrázek 11).



Obrázek 11- Databáze

Při odebírání prvku z databáze je uživatel tázán na potvrzení mazání řádku. Při přidávání nové položky nebo úpravě již vytvořené položky je uživatel přesměrován na novou obrazovku. V horním panelu se nachází tlačítka pro pohyb zpět, potvrzení a zrušení akce (viz Obrázek 12). V hlavním zobrazovacím okně se nachází položky relevantní pro aktálně vybranou tabulku. Pro tabulku zákazníků a dodavatelů jsou informace o názvu a poloze GPS. Poloha musí být vložena v požadovaném formátu, pokud se tomu tak nestane, pole zčervená a nedojde k uložení záznamu. Pro tabulku výbušnin jsou to informace o názvu výrobku, skladovací kód (nebo označení zákazníkovi firmy) a výběr z již zadaných firem do databáze.

Obrázek 12 - Úprava dat

Závěr

Cílem práce bylo vytvoření mobilní čtečky DataMatrix kódu pro operační systém Android. Problémy čtení kódů a jejich katalogizace byly vyřešeny. Aplikace je funkční a připravená pro provoz, po prvních testech aplikace v provozu byla ověřena funkčnost řešení. Výběr knihovny pro čtení kódů se ukázal vhodným. Program je schopný číst a dekódovat DataMatrix kódy v uspokojivém čase, největší nedostatky knihovny Zxing se projevují u čtení kódů nalepených na malých rozbuškách. Pro úspěšné načtení je nutné, aby byl kód nalepen naprosto na rovnou plochu bez záhybů. Tuto problematiku řeší i konkurenční produkty s naprosto stejnými výsledky.

Existující systémy nabízí případnému zákazníkovi přesnější snímací čipy a rychlejší dekódování. Ovšem výsledná aplikace by na tuto konkurenci tvrdě zaútočila cenou, která by byla mnohonásobně nižší. Další výhodou aplikace je konkrétní implementace zaměřená na evidenci kódu výbušnin. Zákazník při používání nabízeného řešení nemusí pořizovat další hardwarové prvky, úplně postačí mobilní telefon s operačním systémem Android.

Evropská unie vydala v dubnu roku 2014 nový dokument, který doporučuje zavedení jednotného formátu označení trhavin. V tomto dokumentu se nachází také doporučení formátu kódu DataMatrix. Toto doporučení zavádí vložení dodatečných informací do kódů. Mezi tyto informace patří informace o původu trhaviny (stát, datum výroby) a informace o trhavině (výrobce, typ produktu, název produktu). Dále se v kódu mají nacházet unikátní čísla trhaviny a další užitečné informace. Tato doporučení mají v dalších letech přejít v povinnost. Výrobci nyní začínají zavádět toto nové označení do výroby. Doporučení o značení trhavin si klade za úkol usnadnit čitelnost kódu pro strojové zpracování. Bohužel v době psaní této práce nebyla tato doporučení v provozu.

Při testování aplikace vyšly na povrch další možnosti na rozšíření aplikace. Mezi případná rozšíření lze zařadit:

- editaci již načtených dat – při načítání může dojít k omylu,
- filtraci dat ve skladu a evidenci – zákazník po krátké době načte velké množství položek, proto by bylo vhodné data filtrovat, a to např. na základě času (dle dní),
- hromadný export – hromadný export dat do PDF,
- zálohování databáze – hromadná záloha celé databáze na webový server nebo do paměti telefonu ve vhodném formátu,
- webové rozhraní – internetové stránky s možností nahlížení na načtená data, jejich export, případné zpřístupnění pro policii,
- implementaci nových pravidel Evropské unie – po této implementaci bude program schopný identifikovat výbušninu a uživatel nebude muset zadávat žádné další informace.

Pro úspěšné uvedení projektu na trh je potřeba implementovat některá z výše uvedených rozšíření. Jako další problém nastává zabezpečení dat. Mezi obchodníky panuje velká rivalita, případný únik dat by mohl vést i k bankrotu. Firma, pro kterou je aplikace vyvíjena, však doufá v rychlé nasazení aplikace do prodeje. S tímto nasazením přichází problém s licencemi a jejich případným obcházením. V dnešní době je jedinou možností kontrolovat licence pomocí webového serveru. Tato varianta však přináší nepříjemnou nutnost připojení k Internetu. Bohužel však nelze aplikaci zabezpečit proti úniku dat i případné krádeže jinak.

Záměr práce byl ve své podstatě splněn. Podařilo se navrhnout plně funkční program, který sice nabízí ještě prostor pro vylepšení, ale také nabízí nemalý prostor pro další inovace, čímž se celý projekt stává i do budoucna zajímavým. Po spuštění testovacího provozu se jako největší přínos ukazuje funkčnost, praktičnost a využitelnost programu v praxi.

Literatura

[1] UJBÁNYAI, Miroslav. Programujeme pro Android. Vyd. 1. Praha: Grada, 2012, 187 s. Průvodce (Grada). ISBN 978-80-247-3995-3.

[2] MURPHY, Mark L. Android 2: průvodce programováním mobilních aplikací. Vyd. 1. Brno: Computer Press, 2011, 375 s. Průvodce (Grada). ISBN 978-80-251-3194-7.

[3] HEROUT, Pavel. Java a XML: průvodce programováním mobilních aplikací. 1. vyd. České Budějovice: Kopp, 2007, 313 s. Průvodce (Grada). ISBN 978-80-7232-307-4.

[4] CHUNG, Fred. Custom Class Loading in Dalvik. *Andorid Delepers Blog* [online]. 2011, 28 JULY 2011 [cit. 2014-05-04]. Dostupné z: <http://android-developers.blogspot.cz/2011/07/custom-class-loading-in-dalvik.html>

[5] Supporting Multiple Screens. GOOGLE. *Android Developers* [online]. 2012? [cit. 2014-05-04]. Dostupné z: developer.android.com/guide/practices/screens_ support.html

[6] I-nigma Camera Phone Barcode Reader SDK. 3GVISION. *3GVision* [online]. ©2013 [cit. 2014-05-04]. Dostupné z: <http://www.i-nigma.com/QR-Barcode-Reader-SDK.html>

[7] OWEN, Sean a Tom TASCHE. Official ZXing ("Zebra Crossing") project home. *GitHub* [online]. 2014, 1-5-2014 [cit. 2014-05-04]. Dostupné z: https://github.com/ zxing/zxing/

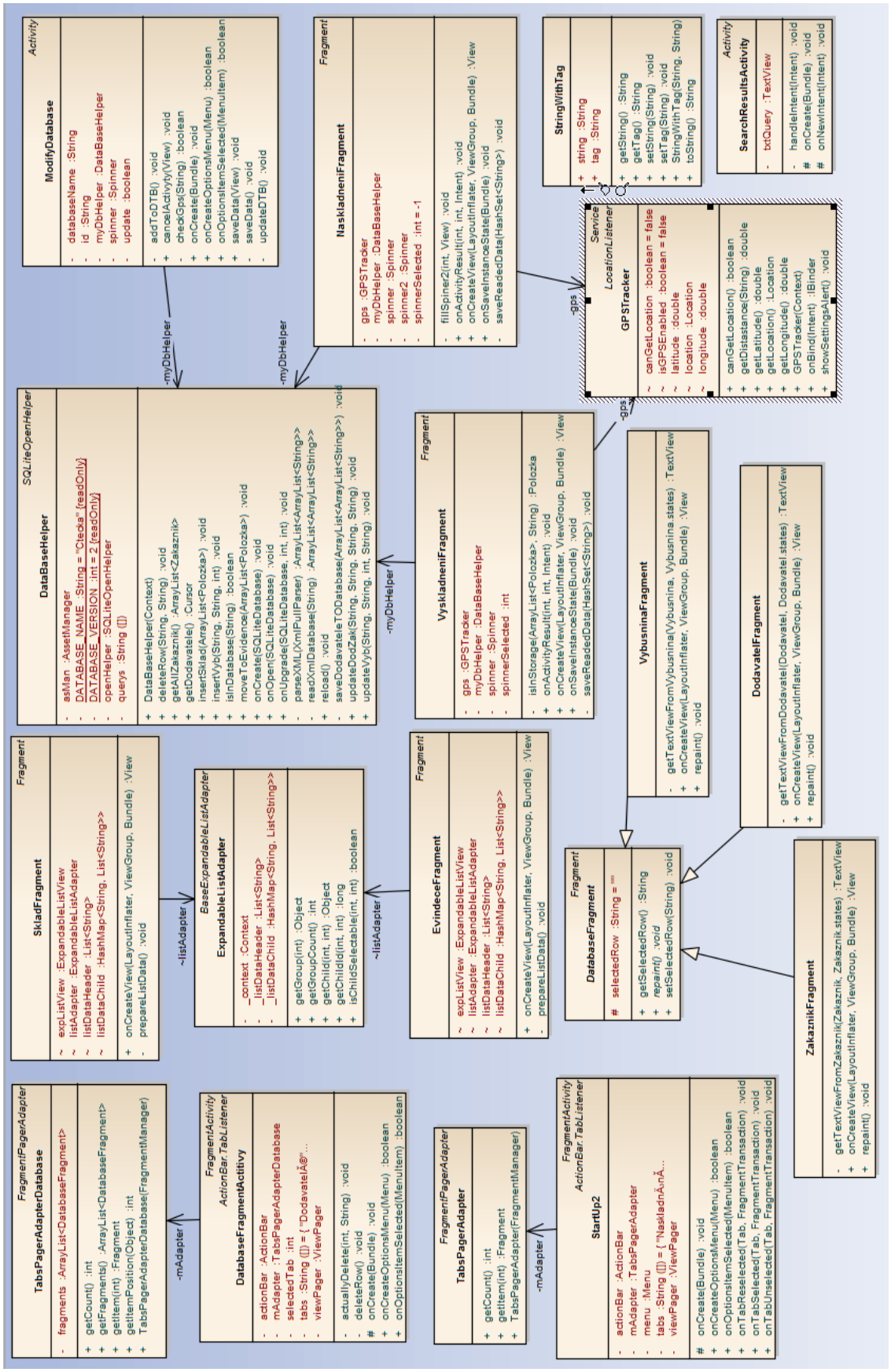
[8] ZBar Support. BROWN, Jeff. *ZBar bar code reader* [online]. 2007-2010, Fri Jul 15 11:19:19 PDT 2011 [cit. 2014-05-04]. Dostupné z: http://zbar.sourceforge.net/ support.html

[9] Android.support.v4.app. GOOGLE. *Android Developers* [online]. 2012? [cit. 2014-05-04]. Dostupné z: http://developer.android.com/reference/android/support/v4 /app/package-summary.html

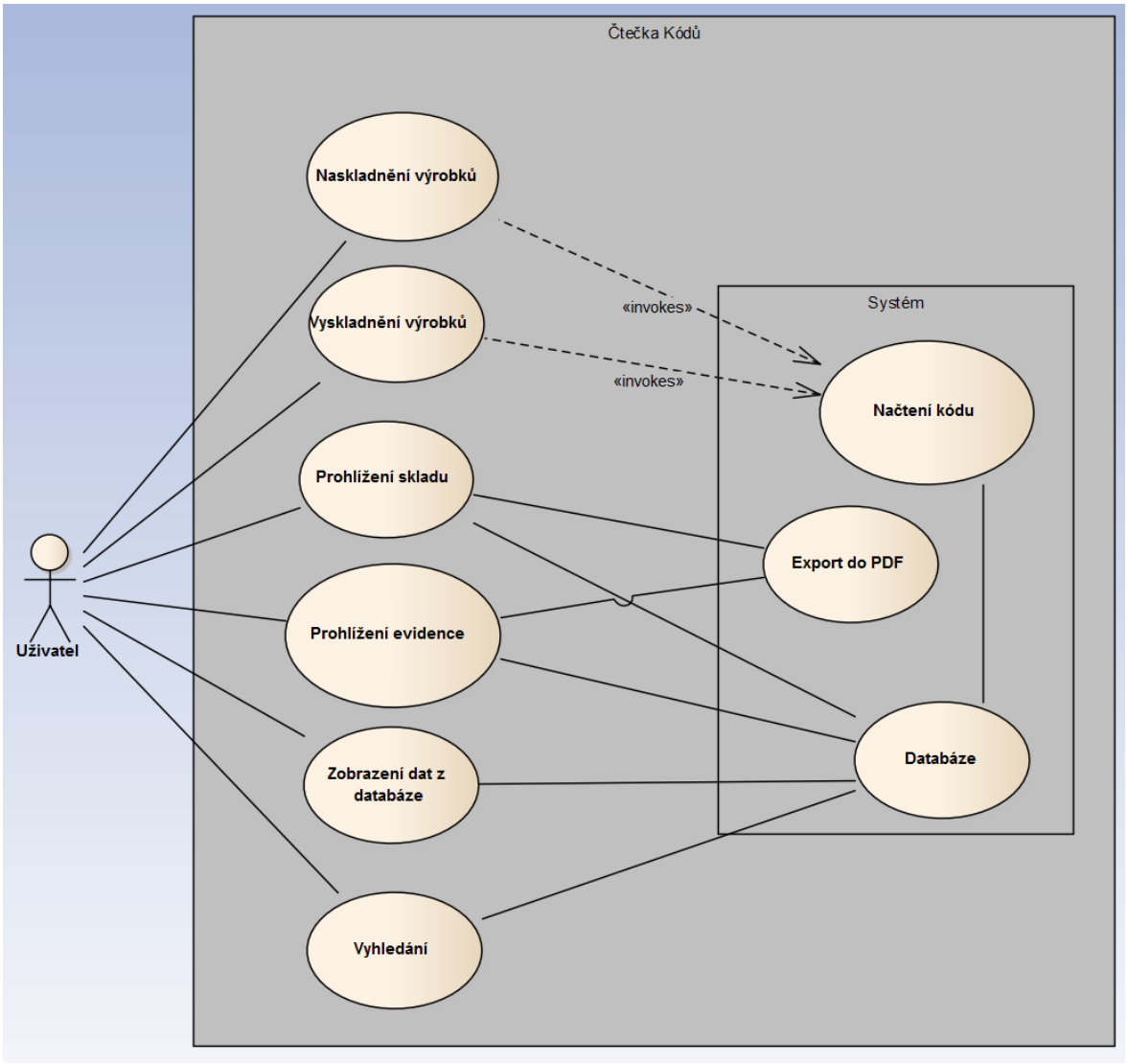
[10] KODYS. *MOTOROLA MC9500* [online]. [cit. 4.5.2014]. Dostupné z: www.kodys.cz /cs/images/content/products/mobilni-terminaly/motorola-mc9500.jpg

[11] GOOGLE. *Action Bar* [online]. [cit. 4.5.2014]. Dostupné z: http://developer.android. com/images/ui/actionbar@2x.png

Příloha A – Diagram tříd



Příloha B – Use Case diagram



Příloha C – Ukázka zdrojového kódu obarvování řádků

```
   TableRow tableRow = (TableRow) view;
   TextView txt = (TextView) tableRow.getChildAt(0);
   String id = txt.getText().toString().trim();
   if (selectedRow.equals(id)) {
       int rank = Integer.parseInt(id);
       if ((rank % 2 != 0))
   tableRow.setBackgroundColor(getResources()
.getColor(android.R.color.background_light));
       else tableRow.setBackgroundColor(getResources().getColor(
android.R.color.darker_gray));
       selectedRow = "";}
   else if (!selectedRow.equals("")) {
       int rank = Integer.parseInt(selectedRow);
       TableLayout tableLayout = (TableLayout) getView().findViewById(
R.id.tab);
       TableRow rowOld = (TableRow) tableLayout.getChildAt(rank - 1);
       if ((rank % 2 != 0)) rowOld.setBackgroundColor(getResources().
getColor(android.R.color.background_light));
                           else rowOld.setBackgroundColor(getResources().
getColor(android.R.color.darker_gray));
view.setBackgroundColor(getResources().getColor(android.R.color.holo_oran
ge_light));
      setSelectedRow(id);
      } else {
view.setBackgroundColor(getResources().getColor(android.R.color.holo_oran
ge_light));
     setSelectedRow(id);
   }
```

Příloha D – Ukázka zdrojového kódu čtení XML souboru

```
   private ArrayList<ArrayList<String>> readXmlDatabase(String xmlFile){
           XmlPullParserFactory pullParserFactory;
           ArrayList<ArrayList<String>> results=null;
           try {
               pullParserFactory = XmlPullParserFactory.newInstance();
               XmlPullParser parser = pullParserFactory.newPullParser();
               InputStream in_s = asMan.open(xmlFile);

parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false);
               parser.setInput(in_s, null);
               results=parseXML(parser);
           } catch (XmlPullParserException e) {
               e.printStackTrace();
           } catch (IOException e) {
               e.printStackTrace();
           }
           return results;
       }
       private ArrayList<ArrayList<String>> parseXML(XmlPullParser
parser) throws XmlPullParserException,IOException {
           ArrayList<ArrayList<String>> products = null;
           int eventType = parser.getEventType();
           ArrayList<String> currentItem = null;

           while (eventType != XmlPullParser.END_DOCUMENT){
               String name = null;
               switch (eventType){
                   case XmlPullParser.START_DOCUMENT:
                       products = new ArrayList<ArrayList<String>>();
                       break;
                   case XmlPullParser.START_TAG:
                       name = parser.getName();
                       if(name.equals("dodavatel")
||name.equals("vybusnina") || name.equals("zakaznik")){
                           currentItem=new ArrayList<String>();
                       } else if(currentItem!=null){
                           currentItem.add(parser.nextText());
                       }
                       break;
                   case XmlPullParser.END_TAG:
                       name = parser.getName();
                       if ((name.equalsIgnoreCase("dodavatel") ||
name.equalsIgnoreCase("vybusnina") || name.equalsIgnoreCase("zakaznik"))
&& currentItem != null){
                           products.add(currentItem);
                       }
               }
               eventType = parser.next();
           }
           return products;
       }
```